# Leveraging data lineage to infer logical relationships between astronomical catalogs

Buddelmeijer, Hugo; Valentijn, Edwin A.

*Published in:*
Experimental Astronomy

*DOI:*
10.1007/s10686-012-9288-z



*Publication date:*
2013

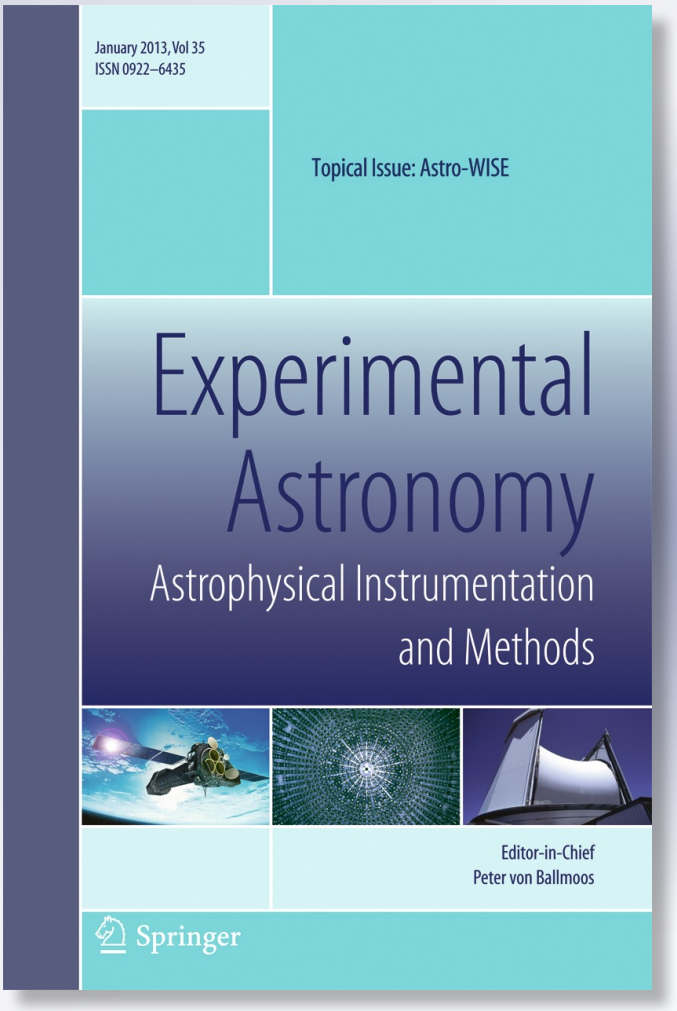
# *Leveraging data lineage to infer logical relationships between astronomical catalogs*

**Hugo Buddelmeijer & Edwin A. Valentijn**

**Experimental Astronomy**
Astrophysical Instrumentation and Methods

ISSN 0922-6435
Volume 35
Combined 1-2

Exp Astron (2013) 35:227-244
DOI 10.1007/s10686-012-9288-z




Springer

Exp Astron (2013) 35:227–244
DOI 10.1007/s10686-012-9288-z

ORIGINAL ARTICLE

# Leveraging data lineage to infer logical relationships between astronomical catalogs

**Hugo Buddelmeijer · Edwin A. Valentijn**

Received: 8 August 2011 / Accepted: 18 January 2012 / Published online: 22 February 2012


**Abstract** A novel method to infer logical relationships between sets is presented. These sets can be any collection of elements, for example astronomical catalogs of celestial objects. The method does not require the contents of the sets to be known explicitly. It combines incomplete knowledge about the relationships between sets to infer a priori unknown relationships. Relationships between sets are represented by sets of *Boolean hypercubes*. This leads to deductive reasoning by application of logical operators to these sets of hypercubes. A pseudo code for an efficient implementation is described. The method is used in the Astro-WISE information system to infer relationships between catalogs of astronomical objects. These catalogs can be very large and, more importantly, their contents do not have to be available at all times. Science products are stored in Astro-WISE with references to other science products from which they are derived, or their dependencies. This creates full *data lineage* that links every science product all the way back to the raw data. Catalogs are created in a way that maximizes knowledge about their relationship with their dependencies. The presented algorithm is used to determine which objects a catalog represents by leveraging this information.

**Keywords** Data mining · Data lineage · Algorithms · Automated theorem solving · Astro-WISE

H. Buddelmeijer (✉) · E. A. Valentijn
Kapteyn Astronomical Institute, P.O. Box 800, 9700 AV, Groningen, The Netherlands
e-mail: buddel@astro.rug.nl

E. A. Valentijn
e-mail: valentyn@astro.rug.nl

## 1 Introduction

A set is a collection of elements. For example, an astronomical catalog is a set with celestial objects as elements. These sets have relationships with one another, for example a set could be a subset of another set. The relationships between sets can be inferred by comparing their elements. However, this is only possible when it is feasible to iterate over all the elements in the sets. A novel method is presented that does not require the contents of the sets to be known explicitly. A priori unknown relationships between sets are inferred by combining incomplete information that is available.

The method is designed for the Astro-WISE information system to infer relationships between astronomical catalogs [1]. Catalog handling using this method is discussed in the following sections. However, the method is generic enough to be used for other purposes.

Catalogs can be stored and even used in Astro-WISE without determining their full contents. The creation of the catalog data is postponed until necessary and the result is only stored when beneficial for performance. That is, the information system will only derive those parts of a catalog that are required for further processing. As a result, the catalog data might not be available as a whole when the catalog is used. One of the key aspects of Astro-WISE is that science products are automatically found or created when requested. This requires the information system to be able to infer the contents of the catalogs automatically. Determining the contents of the catalogs has to be possible without requiring access to the catalog data itself, since this might not be stored.

Astro-WISE stores science products with all the information required to (re)create the data. In particular, every science product is stored with links to other science products from which it was derived, called its *dependencies*. This creates full *data lineage* that links data products all the way back to the raw data. As a result of this, every catalog 'knows' from which other catalogs it is derived. In particular, it is known which relations might hold between the sets of sources of a catalog and those of its dependencies. A priori only this local information about the relationships between catalogs is available. A more global overview of the relationships between catalogs is necessary for the desired automation. The presented method combines this local information to achieve the required knowledge.

The novelty of the method is the use of Boolean hypercubes to represent relations between sets. Relationships between specific sets are represented as sets of hypercubes in order to account for incomplete knowledge. This makes it possible to deduce relationships by application of logical operators on these sets of hypercubes.

Ultimately, the presented method is a specialized form of automated theorem proving. Other such methods could be used to infer relationships, for example software that can solve the problems in the SET domain of the TPTP

Problem Library[1] [7]. Those methods are very generic and can be used to solve several kinds of logical problems. The presented mechanism is more specific: while the used hypercube representation is natural for dealing with sets, it is not directly applicable to more general problems.

Relational databases can use similar mechanisms for query optimization (see for example [2] for an overview). These are embedded in the optimization algorithms and are therefore not directly applicable to the requirements of Astro-WISE.

This paper is structured as follows. The representation of relationships by means of sets of hypercubes and the details of the method are given in Section 2. Applications of the presented mechanisms in Astro-WISE are discussed in Section 3. Subsequently, the pseudocode of the algorithms is given in Section 4. This is followed by an example in Section 5 and conclusions in Section 6.

## 2 Description of algorithms

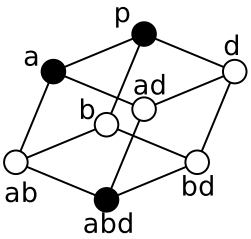
The basis of the method is the use of Boolean hypercubes to represent logical relations between sets (Section 2.1). The relationships between specific sets are represented by means of a set of hypercubes to account for incomplete knowledge (Section 2.2). Deduction is possible through application of logical operators on the sets of hypercubes (Section 2.3). Scalability in implementation is achieved by optimizing important logical operators (Section 2.4). Pseudo code for an implementation of the method is given in Section 4.

### 2.1 Relations as hypercubes

A Boolean hypercube can be used to represent a relation between sets by associating a set to each dimension of the hypercube (Fig. 1a). This representation is well suited for a numerical implementation by means of a multidimensional array (Fig. 1b). Examples of hypercube representations of relations are given in Table 1. In particular the relations that are directly relevant to our astronomical application are shown.

Every logical relation between $n$ sets can be represented by means of an $n$-dimensional hypercube. This is done by identifying each of the $2^n$ possible intersections between the sets with one of the vertices of the hypercube. A vertex in the second position of a specific dimension represents objects that are elements of the set corresponding to that dimension. A vertex in the first position of the dimension represents objects not in the corresponding set. For example, the vertex that is in the second position in all dimensions represents objects that are in all the sets described by the hypercube. The vertex that is

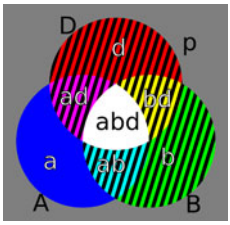
[1] http://www.cs.miami.edu/~tptp/




(a) Hypercube

```
[[[ True, False],
  [False, False]],
 [[ True, False],
  [ False,  True]]]
```

(b) Array




(c) Shaded Venn

**Fig. 1** Hypercube, array and shaded Venn representation of the relationship between sets $A$, $B$ and $D$. The vertices represent intersections between sets as indicated with *lower case letters*. The vertex labeled $p$ represents objects not in any set under consideration. *Solid vertices* indicate that the corresponding intersection is non-empty. The array representation can be used verbatim in the Python programming language by means of the numpy package. None of the sets are empty, set $B$ and $D$ are equal and set $A$ is a superset of them, but does not contain all objects in the universe

**Table 1** Examples of hypercube representations of low dimensional relations

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | 0D |
| | | | | | | | | | | | | 1D |
| ✓ | | ✓ | | | | | | | | | | empty |
| | (3) | | | | (3) | | | | | | | 2D |
| ✓ | ✓<br>✓✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | equality<br>subset<br>superset |
| | (3) | | | | (3) | | | | | | | 3D union |
| | (3) | | | | (3) | | | | | | | intersec. |

Relations that are not directly relevant to our astronomical application are omitted. They are replaced with a number in parenthesis that indicates the number of missing hypercubes. The top vertex of each hypercube represents objects not in any of the sets under consideration, the bottom vertex objects in all the sets. A *solid circle* is used for `True` values and an open circle for `False`. The hypercubes are ordered hierarchically: a relation in a lower cell implies the relations of lower dimension in the cells above it. The *check marks* at one and two dimensions indicate that a set of these hypercubes represents the relation mentioned in the last column. Check marks below the numbers in parenthesis indicate the number of omitted hypercubes that are part of this set. The *superset* and *subset* labels in the 2D rows refer to the extra dimension with respect to the one already present in the 1D row. Furthermore they are *strict*: an equality is not considered a subset. Of the 256 three dimensional relations, only those where the third dimension corresponds to the union or intersection of the first two are shown

in the first position in all dimensions represents objects that are in none of the sets under consideration. A Boolean value can be assigned to each vertex to indicate whether the corresponding intersection between sets contains any objects: a Boolean `True` value is assigned if the vertex represents one or more objects and a Boolean `False` value is assigned if it does not. The collection of all objects—whether inside a set or not—is called the *universe*, which can be empty.

This hypercube representation of relations between sets is similar to Karnaugh maps [5] and to the hypercube representation of logical operators by [3]. Furthermore, the hypercubes can be translated into shaded Venn diagrams [8] by assigning every vertex to a region of overlap in the Venn diagram.

A hypercube of a certain dimension also represents specific relations of lower dimensions. A lower dimensional relation is inferred by summing the hypercube over the dimensions that represent the unwanted sets (Algorithm 1). Summing over a dimension means repeatedly performing the logical *or* operator on two adjacent vertices that are aligned in that dimension, since Boolean values are assigned to the vertices.

**Algorithm 1** Removing a set from a relationship

**Require:** $H_o$ = set of hypercubes of the original relationship
**Require:** $d$ = dimension corresponding to the to-be-removed set.
**Ensure:** $H_r$ = set of all hypercubes consistent with $H_o$ without dimension $v_d$

1: $n \leftarrow$ dimension of the hypercubes in $H_o$
2: $H_r \leftarrow$ empty set of hypercubes
3: **for all** hypercubes $h_o$ in $H_o$ **do**
4: $h_r \leftarrow$ hypercube of dimension $n-1$ with all values set to `False`
5: **for all** cells $c_r = h_r(v_1, ..., v_{d-1}, v_{d+1}, ..., v_n)$ in $h_r$ **do**
6: $c_p \leftarrow h_o(v_1, ..., v_{d-1}, 0, v_{d+1}, ..., v_n)$
7: $c_q \leftarrow h_o(v_1, ..., v_{d-1}, 1, v_{d+1}, ..., v_n)$
8: $h_r(v_1, ..., v_{d-1}, v_{d+1}, ..., vn) \leftarrow c_p \vee c_q$
9: **end for**
10: $H_r \leftarrow H_r \cup \text{set}(h_r)$
11: **end for**

## 2.2 Relationships as sets of hypercubes

A relationship between specific sets is described with a set of all hypercubes that are consistent with the available knowledge about the relationship. This stems from our astronomical requirements, where the exact relationship between sets is not always known. For example, there are four different hypercubes that represent an equality: between empty or nonempty sets and with or without objects outside the considered sets (Table 1). Representing that two sets are identical, without any extra available information, should therefore be done with a set of these four hypercubes. However, more information is

usually not necessary: it is enough to determine that the relationship between two sets must be one of these four in order to infer that they are equal.

The set of hypercubes representation allows us to define four classes of relationships between sets:

- The *Contradiction*, an empty set of hypercubes: there is no relation between the sets that is consistent with the available knowledge.
- An *Exact Relation*, a set with exactly one hypercube: there is only one relationship possible between the sets; everything is known about the sets under consideration.
- An *Inexact Relation*, a set with more than one hypercube: there are several relations that are consistent with the available knowledge.
- The *Tautology*, a set with all $2^{2^n}$ possible hypercubes representing $n$ sets: every relationship is possible; nothing is known about these sets.

The use of the tautology can be prevented in a numerical implementation (Section 2.4). It is included here because it is useful in discussing the presented mechanisms.

A relationship also represents knowledge about sets that do not correspond to any dimension of the hypercubes. For example, an empty universe can be represented with a hypercube of any dimension with `False` as the value of all vertices. Such a relation implies that all sets, also those that have no corresponding dimension in the hypercube, must be empty. Most relationships are less strict: in general they tend to represent the tautology for sets that have no corresponding dimension.

A higher dimensional relationship can be inferred from a lower dimensional one by adding dimensions to the hypercubes. This results in a set of hypercubes that can be constructed as follows (Algorithm 2): first create the tautology for the higher dimension. Subsequently remove all hypercubes that are not consistent with the original relationship. Adding a set to a relationship usually results in an increase of the number of hypercubes necessary to represent the relationship.

**Algorithm 2** Adding a set to a relationship in a naive way

**Require:** $H_o$ = set of hypercubes of the original relationship
**Ensure:** $H_r$ = set of all hypercubes consistent with $H_o$ with one extra dimension

1: $n \leftarrow$ dimension of the hypercubes in $H_o$
2: $H_r \leftarrow$ empty set of hypercubes
3: $H_t \leftarrow$ set of all $2^{(2^{n+1})}$ distinct hypercubes of dimension $n+1$
4: **for all** hypercubes $h_t$ in $H_t$ **do**
5: $\quad h_u \leftarrow h_t$ with dimension $v_{n+1}$ removed (Algorithm 1)
6: $\quad$ **if** $h_u$ in $H_o$ **then**
7: $\quad\quad H_r \leftarrow H_r \cup \text{set}(h_t)$
8: $\quad$ **end if**
9: **end for**

### 2.3 Logical operations on relations

A natural way to apply logical operators to relations follows from the use of sets of hypercubes to represent the relations. The basic principle is that applying a logical operator to one or more relations, amounts to applying this operator to their corresponding sets of hypercubes. This leads to an implicit way to infer unknown relationships from known ones by application of the material (non)implication.

The only non-trivial unitary operator is the negation (NOT, $\neg$). The negation of a relationship between $n$ sets is represented by the set of hypercubes of dimension $n$ that are not consistent with the original relationship. This set can be constructed by creating the tautology of dimension $n$ and removing those hypercubes that were used to represent the original relationship (Algorithm 3). This is not scalable, because the size of the tautology grows exponentially with the number of dimensions. The negation should therefore be avoided, and thereby also its implicit use in binary operators.

**Algorithm 3** Applying the Negation operator: $r = \neg q$

**Require:** $H_p$ = set of hypercubes of the original relationship
**Ensure:** $H_r$ = set of all hypercubes of the same dimension that are not consistent with $H_p$

```
1: n ← dimension of the hypercubes in H_p
2: H_r ← empty set of hypercubes
3: H_t ← set of all 2^(2^n) distinct hypercubes of dimension n
4: for all hypercubes h_t in H_t do
5:    if not h_t in H_p then
6:       H_r ← H_r ∪ set(h_t)
7:    end if
8: end for
```

Applying a binary operator to two relationships requires that the used hypercubes represent the same sets (Algorithm 4). In general this can be achieved by adding sets to each relationship (through Algorithm 2) until they represent relationships between identical sets. However, in some cases it suffices to remove sets from the relationships (Section 2.4). Binary operators that are of particular importance for the deduction of a priori unknown relationships are:

- Conjunction (AND, $\wedge$, $\cap$): Combines two relationships that are both known to hold. The result of $P \wedge Q$ is a relationship represented by hypercubes that are consistent with both a hypercube in $P$ and one in $Q$.
- Disjunction (OR, $\vee$, $\cup$): Combines relationships of which it is known that at least one of them holds. The result of $P \vee Q$ is a relationship represented by hypercubes that are consistent with a hypercube in $P$ and/or one in $Q$.
- Material Implication ($\rightarrow$): Can be used to infer relations. The result of $P \rightarrow Q$ is a relationship with hypercubes that are consistent with both $P$ and $Q$, together with those that are not consistent with $P$. The relationship

$P$ implies that relationship $Q$ holds when $P \rightarrow Q$ results in the tautology. The material implication $(P \rightarrow Q)$ can be implemented as $(\neg(P \wedge (\neg Q)))$, which requires the negation operator. An implementation of the negation is not scalable; the material implication is therefore not suitable to prove whether unknown relations hold.

– Material Nonimplication ($\nrightarrow$): Can also be used to infer relationships. The relation that is the result of the material nonimplication $(P \nrightarrow Q)$ is represented by the set of hypercubes that is consistent with $P$, but not with $Q$. This operation can be used to prove that relation $Q$ must hold given $P$, because $P$ implies $Q$ when the result of the operation is the contradiction. This operation is more suitable for implementation than the material implication, because it always results in a relation that is represented by less hypercubes than the original relations.

**Algorithm 4** Applying any binary operator: $r = p \circ q$

**Require:** $H_p$ = set of hypercubes of the first relationship
**Require:** $\Lambda_p$ = list of labels that correlates sets to the dimensions of $H_p$
**Require:** $H_q$ = set of hypercubes of the second relationship
**Require:** $\Lambda_q$ = list of labels that correlates sets to the dimensions of $H_q$
**Require:** $\circ$ = the logical operation to be applied to the relationship
**Ensure:** $H_r$ = set of hypercubes that is consistent with both $H_p$ and $H_q$

```
for all labels λ_p in Λ_p not in Λ_q do
    H_v ← H_q with dimension λ_p added (Algorithm 2, 6)
end for
for all labels λ_q in Λ_q not in Λ_p do
    H_u ← H_p with dimension λ_q added (Algorithm 2, 6)
end for
H_r ← H_u ∘ H_v
```

The logical operators can be used to prove that a specific relationship must hold by testing for entailment (Algorithm 5). First a list of relationships $(S_0, S_1, ...)$ is constructed, where each $S_i$ contains partial a priori knowledge

**Algorithm 5** Testing for entailment

**Require:** $P_i$ = set of relationships representing the a priori knowledge, $i = 1, 2, ...$
**Require:** $Q$ = a relation for which it is unknown whether it holds
**Ensure:** $r =$ `True` when $P_i$ entails $Q$, `False` otherwise

```
P ← (P_0 ∧ (P_1 ∧ ...))
R ← (P ↛ Q)
if R == Contradiction then
    r ← True
else
    r ← False
end if
```

about the sets. The logical conjunction operator is subsequently applied to all these relationships, resulting in relationship $S$. Finally, the nonimplication $S \nrightarrow R$ is applied, where $R$ is the relationship that needs to be proven. Relationship $R$ must be valid if the result of the nonimplication is the contradiction.

### 2.4 Optimizations

The logical operators are discussed in the previous section in an intuitive but naive form that will lead to an unscalable implementation. Firstly, adding a set to a relationship requires the creation of all hypercubes of a specific dimension. This is not feasible for dimensions higher than about 4. This can be avoided by not creating hypercubes with `True` vertices that correspond to two `False` vertices in the original (Algorithm 6). Secondly, enlarging the set of hypercubes in order to apply binary operators can sometimes be avoided entirely, in particular for conjunction and material nonimplication.

**Algorithm 6** Improved way of adding a set to a relation

**Require:** $H_o$ = set of hypercubes of the original relation
**Ensure:** $H_r$ = set of all hypercubes consistent with $H_o$ with one extra dimension

1: $n \leftarrow$ dimension of the hypercubes in $H_o$
2: $H_r \leftarrow$ empty set of hypercubes
3: **for all** hypercubes $h_o$ in $H_o$ **do**
4: $h_t \leftarrow$ hypercube of dimension $n+1$ with all values set to `False`
5: **for all** cells $c_o = h_o(v_1, v_2, ..., v_n)$ in $h_o$ **do**
6: $h_t(v_1, v_2, ..., v_n, 0) \leftarrow c_o$
7: $h_t(v_1, v_2, ..., v_n, 1) \leftarrow c_o$
8: **end for**
9: $H_t \leftarrow \text{set}(h_t)$
10: **for all** cells $c_o = h_o(v_1, v_2, ..., v_n)$ in $h_o$ **do**
11: **if** $c_o$ == `True` **then**
12: **for all** hypercubes $h_t$ in $H_t$ **do**
13: $h_u \leftarrow h_t$
14: $h_u(v_1, v_2, ..., v_n, 0) \leftarrow$ `False`
15: $h_v \leftarrow h_t$
16: $h_v(v_1, v_2, ..., v_n, 1) \leftarrow$ `False`
17: $H_t \leftarrow H_t \cup \text{set}(h_u, h_v)$
18: **end for**
19: **end if**
20: **end for**
21: $H_r \leftarrow H_r \cup H_t$
22: **end for**

Adding sets to $P$ with the purpose of performing the conjunction $P \wedge Q$ can often be done without enlarging the number of hypercubes. This is the case when for each hypercube of $P$ there is at most one more-dimensional

hypercube that is consistent with both $P$ and $Q$. Algorithm 7 shows how to verify this condition when only one of the sets in $Q$ is not in $P$. The algorithm checks for a one-to-one correspondence between the hypercubes of $Q$ and the hypercubes of $Q$ with this extra set removed. This correspondence, if existent, can be used to add the extra set to $P$ without enlarging the number of hypercubes.

**Algorithm 7** Enhanced conjunction: $r = p \wedge q$

**Require:** $H_p$ = set of hypercubes of dimension $n_p$ of the original relation
**Require:** $H_q$ = set of hypercubes of dimension $n_q$ where only the last dimension $\nu_{n_q}$ is not present in $H_p$ and the first dimensions $\nu_1$ to $\nu_{n_q-1}$ correspond to the first dimensions of $H_p$
**Ensure:** $H_r$ = set of hypercubes of dimension $n_p + 1$ that is consistent with both $H_p$ and $H_q$

$H_t \leftarrow H_q$ with dimension $n_q$ removed (Algorithm 1)
**if** length($H_t$) == length($H_q$) **then**
  $h_t \leftarrow$ hypercube of dimension $n_q$ with all values set to `False`
  **for all** hypercubes $h_q$ in $H_q$ **do**
    **for all** cells $c_t = h_t(\nu_1, \nu_2, ..., \nu_{n_q})$ in $h_t$ **do**
      $h_t(\nu_1, \nu_2, ..., \nu_{n_q}) \leftarrow c_t \vee h_q(\nu_1, \nu_2, ..., \nu_{n_q})$
    **end for**
  **end for**
  $H_r \leftarrow$ empty set of hypercubes
  **for all** hypercubes $h_p$ in $H_p$ **do**
    $h_r \leftarrow$ hypercube of dimension $n_p + 1$ with all values set to `False`
    **for all** cells $c_p = h_p(\nu_1, \nu_2, ..., \nu_{n_p})$ in $h_p$ **do**
      **if** $c_p$ == *True* **then**
        **if** $h_t(\nu_1, \nu_2, ..., \nu_{n_q-1}, 0)$ == *True* **then**
          $h_r(\nu_1, \nu_2, ..., \nu_{n_p}, 0) \leftarrow$ *True*
        **else**
          $h_r(\nu_1, \nu_2, ..., \nu_{n_p}, 1) \leftarrow$ *True*
        **end if**
      **end if**
    **end for**
    $h_m \leftarrow h_r$ with dimension $n_p + 1$ removed
    **if** $h_m$ in $H_p$ **then**
      $H_r \leftarrow H_r \cup$ set($h_r$)
    **end if**
  **end for**
**else**
  Create $H_r$ through the regular conjunction with Algorithm 4.
**end if**

The material nonimplication operator can sometimes be performed by *removing* dimensions instead of adding them, because it tests for inconsistency (Algorithm 8). This is possible for the operation $P \nrightarrow Q$ when all the sets of

$Q$ are also represented by $P$. It is not necessary to add the extra dimensions of $P$ to $Q$ in order to test which hypercubes in $P$ are inconsistent with $Q$: the hypercubes of $Q$ essentially represent the tautology for these extra sets and it is not possible to be inconsistent with the tautology. Instead, the extra dimensions can be removed from the hypercubes of $P$ to determine whether the originals are consistent with $Q$.

**Algorithm 8** Enhanced material nonimplication: $r = p \nrightarrow q$

**Require:** $H_p$ = set of hypercubes of the first original relation
**Require:** $\Lambda_p$ = list of labels that correlates sets to the dimensions of $H_p$
**Require:** $H_q$ = set of hypercubes of the second original relation
**Require:** $\Lambda_q$ = list of labels that correlates sets to the dimensions of $H_q$
**Require:** $\text{set}(\Lambda_q) \subseteq \text{set}(\Lambda_p)$
**Ensure:** $H_r$ = set of hypercubes that is consistent with $H_p$ but not with $H_q$
  $H_r \leftarrow$ empty set of hypercubes
  **for all** hypercubes $h_p$ in $H_p$ **do**
    $h_t \leftarrow h_p$
    **for all** $\lambda_t$ in $\Lambda_p$ not in $\Lambda_q$ **do**
      $h_t \leftarrow h_t$ with dimension corresponding to $\lambda_t$ removed (Algorithm 1)
    **end for**
    **if** $h_t$ in $H_q$ **then**
      $H_r \leftarrow H_r \cup \text{set}(h_p)$
    **end if**
  **end for**

Furthermore, sets that are equal can be represented with the same dimension of the hypercubes. This optimization would make the presentation of the algorithms more complicated without adding conceptual insights and is therefore not discussed in this paper.

## 3 Astro-WISE

The presented method is used in the Astro-WISE information system to handle astronomical catalogs. These catalogs contain information about astronomical objects and can therefore be seen as sets with these objects as elements. Catalogs in Astro-WISE are primarily created either from images or by performing an operation on other catalogs; the mechanisms presented in this paper are only used for the latter kind.

### 3.1 Objects and dependency graphs

Astro-WISE uses an Object-Oriented data model in which science products are stored as class instantiations. Every class forms a blueprint of how its instances should be processed to create the data from other objects. Every object has *persistent properties* that are stored in the database, which allows the

object to be used across sessions and shared between scientists. The persistent properties of an object include all the details of its processing: its *dependencies*, and the values of any process parameters that can be set. Different catalog classes are designed for different operations to create catalogs.

To create full data lineage, the depencies of an object have their own dependencies. This net of dependencies that links an object to the raw data is called a *dependency graph* (Fig. 2). The algorithms presented in this paper are used for the automatic creation and manipulation of dependency graphs dealing with catalogs.

### 3.2 Target processing

The heart of Astro-WISE is its request driven way of data handling, called *target processing* [6]. In the traditional way of data handling, scientists start with a data set and perform operations until they reach their required end product. Target processing turns this around: scientists request the desired end product directly—their *target*—and the information system will create a dependency graph that ends with an object representing the requested data. The information system can reuse existing objects, possibly created by other scientists. Furthermore, it can autonomously create new objects, because the class definition forms a blueprint for new objects.

The data lineage allows any object to be processed at any time, because the object's class and persistent properties describe how this can be done. This is taken to the extreme for catalog instances: catalogs can be created and

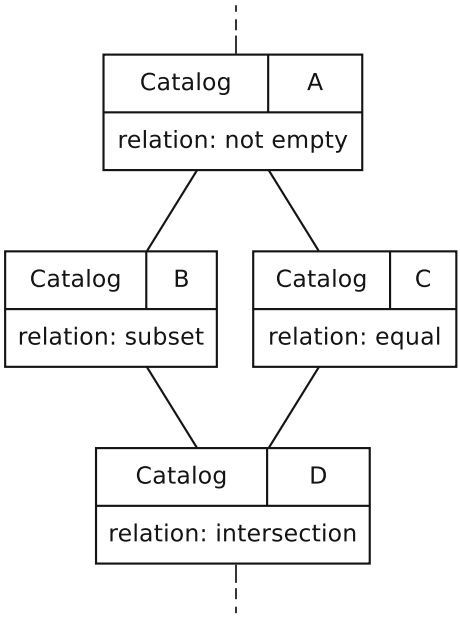



**Fig. 2** A simplified part of a dependency graph in Astro-WISE. Catalog *A* contains a known set of sources. Catalog *B* represents an yet unknown subset of the sources of *A*. Catalog *C* represents the same sources as *A* with a different set of attributes. Catalog *D* represents an intersection of the sources of *B* and *C* with the attributes of both *B* and *C*

stored without fully processing them, or without processing them at all [1]. In other words, it is not required to create or store the contents of a catalog as a whole, achieving the scalability required to handle large catalogs. Therefore, determining the contents of the catalogs should be possible without consulting the catalog data directly.

The information system can process catalogs partially by modification of dependency graphs. This allows new catalogs to be created in their most general form to maximize their reusability for future requests. At the same time this ensures that catalog data is only created when this is essential for the requested dataset. Optimization of the dependency graphs requires the information system to know as much as possible about the contents of the catalogs in the graph *before* they are processed.

### 3.3 Algorithm specifics

The presented method determines whether a desired relation holds by combining information about known relationships between catalogs. The catalog classes for Astro-WISE are designed to maximize this a priori knowledge. In particular, every catalog class corresponds to a specific operation to derive catalog data from other objects. Many of these correspond to relational operators [4]. Each catalog class allows only a specific set of relationships between the sets of sources of a catalog and its dependencies.

Every catalog instance has partial knowledge about its relationship with its dependencies: it knows what relations are permitted by its class, not which of those actually holds. A priori this is the only available information. The presented mechanism is used to acquire knowledge that requires combining this local information.

In this astronomical setting, sets correspond to astronomical catalogs, and the elements are astronomical objects. This background puts several constraints on the use of the algorithm:

- All catalogs by design have one of the following relationships with their dependencies, with in brackets the number of corresponding hypercubes (Table 1): equality (4), subset (4), intersection (16) or union (16). However, they can have any relationship with catalogs that are not their direct dependency.
- The following relations are the most important in checking which relations hold between sets: non-emptiness (2), equality (4), superset (4).
- A relation where all objects are within a set can never hold.

Most of the relations that are enforced by the catalog classes are shown in Table 1. In Section 5 the method is applied to a simplified Astro-WISE use case.

### 3.4 Scalability

The major factor in the scalability of the method is the size of the set of hypercubes used to represent relationships. Adding a dimension to a hypercube, will

result in $3^t$ new hypercubes, where $t$ is the number of `True` cells in the original hypercube. The size of the hypercubes themselves is less of an issue: these scale with $2^n$, while the number of possible hypercubes scales with $2^{2^n}$ where $n$ is the number of sets. The number of possible hypercubes can grow very rapidly with the number of sets when little is known about their relationship. However, this is not necessarily problematic for application in Astro-WISE:

- Many catalogs represent the exact same objects. It is not required to add a new dimension to the hypercubes when adding a set that is known to be equal to one of the other sets: the set can be associated with an existing dimension.
- Sets that are different can still have a relation that is quantified by a low number of `True` cells. For example, sets that are a subset of another set occur often and require only one extra `True` cell. Furthermore, some sets, e.g. those that are the intersection of sets already in a relation, can be added without increasing the number of `True` cells at all (Algorithm 7). The relations that require the most `True` cells, such as disjoint or partially overlapping sets, are rare, because comparisons are done on catalogs that are connected through data lineage.
- Some relations are very unlikely to occur at all. For example there will always be objects not in any set.

Nonetheless, the set of hypercubes can become large for large dependency graphs of catalogs. However, in most cases the number of hypercubes can be limited:

- External knowledge—with respect to this algorithm—can be used explicitly. For example it can often be determined whether a catalog is empty or whether two catalogs are disjoint.
- Any knowledge about the relationships that is obtained, through the algorithm or otherwise, can be stored for future use.
- The sets of hypercubes are created by traversing the dependency graphs of catalogs. The most interesting relationships in a dependency graph are those between the begin and end points. Dimensions that correspond to catalogs in the middle of a dependency graph might be removed when this has little or no influence on the relationships between the catalogs at the edges.

This combination of factors ensures that the algorithm is sufficiently scalable to meet the requirements for use in Astro-WISE.

## 4 Pseudocode

The pseudocode for the algorithms mentioned above is presented. Every relationship $P$ is assumed to be represented with a set of hypercubes $H_p$ and a set of labels $\Lambda_p$. These labels identify the dimensions of the hypercubes with the sets considered by the relationship. The administration of these

labels is trivial and is therefore only discussed when relevant for handling the hypercubes.

Dimensions of the hypercubes are denoted with $\nu$'s. A specific vertex or cell in a hypercube $h_p \epsilon H_p$ is denoted with $h_p(\nu_1, \nu_2, ..., \nu_n)$, where each $\nu_i$ can have a value of 0 or 1. It is assumed that the dimensions of the hypercubes are in the same order when they are compared. A transposition of the dimensions suffices to accomplish this when necessary.

## 5 Example

The method is demonstrated with a simplified application in Astro-WISE. We will use the terms *source* for an astronomical object in a catalog, that is, an element in a set. Furthermore we use the term *attribute* for a quantified physical property of that object, for example its mass. Figure 2 shows a simplified part of a dependency graph consisting of four catalogs:

- Catalog $A$ is the base catalog from which the others are derived and contains a finite, known, set of sources. The catalog does not contain all the sources in the Universe.
- Catalog $B$ represents a subset of the sources of $A$. The selection criterion is known, but unevaluated. The contents of $B$ is therefore unknown, and it might even be empty.
- Catalog $C$ represents new attributes of the sources in catalog $A$. That is, the attributes are not in catalog $A$ and have to be derived. The values of these attributes do not have to be calculated or stored in order to create the dependency graph. Catalog $C$ represents the same sources as catalog $A$.
- Catalog $D$ combines the attributes of catalogs $B$ and $C$ and represents an intersection of their sources. The precise contents of this catalog is unknown at its creation, because the selection criterion of $B$ is not yet evaluated and the attributes of $C$ are not yet calculated.

Such a dependency tree would have been created automatically by requesting the attributes from $A$ and $C$ for the sources specified in $B$. The information system will attempt to process this dependency graph in an optimal way. In this case, it will try to limit the processing of catalog $C$ to those sources that are required to process $D$. A priori, the only available information about $D$ is the local knowledge that $D$ has about its relationship with $B$ and $C$. The algorithm is applied to determine that set $D$ represents the exact same sources as $B$. The following steps are performed, visualized in Fig. 3:

- All the hypercubes consistent with the local information are created as relationships $A$ (nonempty), $AB$ (subset), $AC$ (equality) and $BCD$ (intersection).

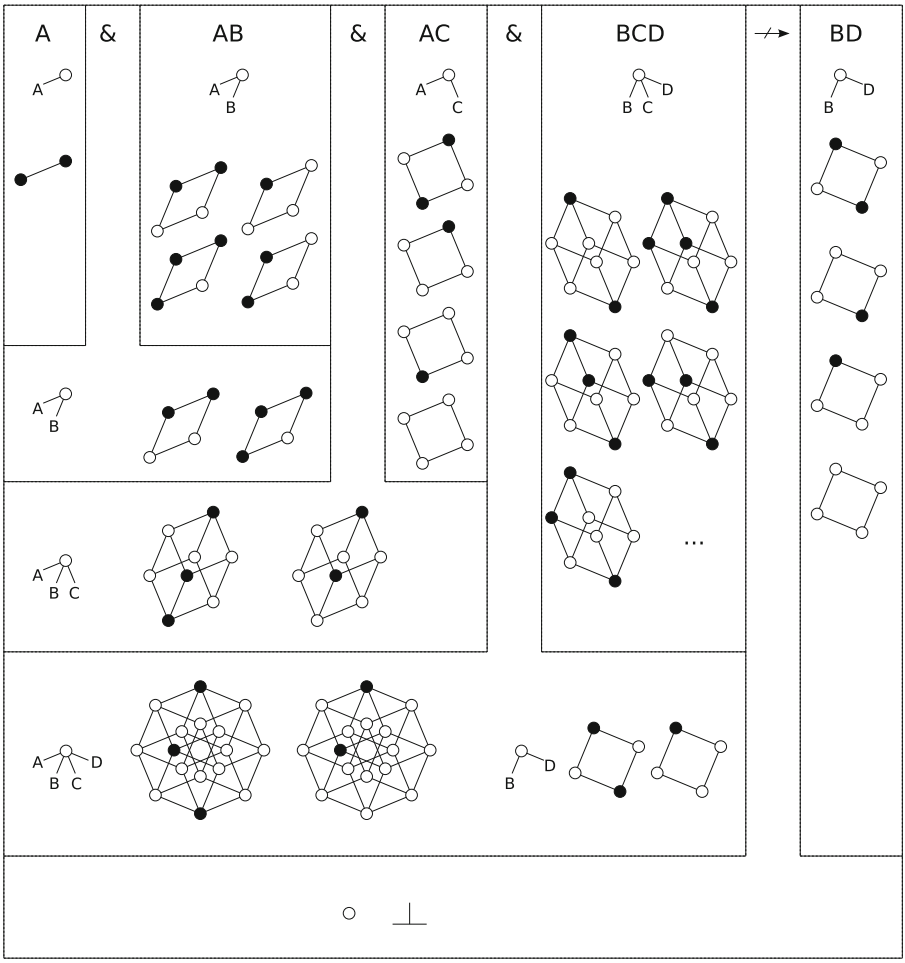



**Fig. 3** The determination that catalogs $B$ and $D$ of Section 5 (Fig. 2) represent the same astronomical objects. The hypercubes are projected such that a specific set is always oriented in the same direction. A *solid circle* represents a `True` value and an open circle a `False` one. The *circle at the top* represents the first cell in all dimensions. The four dimensional hypercubes of the relationship between $A$, $B$, $C$ and $D$ are also shown in reduced form as a relationship between $B$ and $D$. $B$ must be equal to $D$ because both hypercubes representing the relationship between $A$, $B$, $C$ and $D$ are consistent with this equality

- The conjunction operator is subsequently applied on these relationships. Dimensions are added to the hypercubes when necessary. The result is a four dimensional relationship between $A$, $B$, $C$ and $D$.
- Relationship $BD$ is created, representing an equality between $B$ and $D$. It is a priori unknown whether this relation holds.
- The material nonimplication operator is applied to relation $ABCD$ and $BD$, resulting in the Contradiction (Section 2.2). That is, there are no possible relationships between $A$, $B$, $C$ and $D$—given the a priori

knowledge—in which $B$ and $D$ are not equal. Therefore, $B$ and $D$ must represent the same sources.

The information system will optimize the dependency graph using the knowledge that the sources in $B$ and $D$ are equal. In particular, it will evaluate the selection criterion in $B$ and only calculate the attributes of $C$ for those sources. The conclusion that $B$ is equal to $D$ is reached without having to consult any catalog data, which was necessary because the catalog data had not yet been created.

## 6 Conclusions

A novel mechanism for inferring relationships between sets is discussed. It is shown that the use of sets of hypercubes to represent relationships leads to a natural way of inferring a priori unknown relations: deduction is performed by combining incomplete knowledge through the application of logical operators. Algorithms that are suitable for a scalable implementation are presented, including pseudocode.

The novel aspects of the method were demonstrated by its use in Astro-WISE, where the sets correspond to catalogs and the elements to astronomical objects. Catalogs can be stored and used in Astro-WISE without their content being evaluated. The method is used to acquire knowledge about their contents without requiring direct access to the catalog data. This has lead to design choices in the way catalogs are handled in Astro-WISE: catalogs are created such that the knowledge about their relationships is maximized.

An automated way to infer relations between catalogs is essential for the request driven way of processing in information systems such as Astro-WISE. The presented algorithms form an excellent method to accomplish this. The method is generic enough to be implemented in any programming language and can be used by any information system.

**Acknowledgements** This research is part of the project “Astrovis”, research program STARE (STAR E-Science), funded by the Dutch National Science Foundation (NWO), project no. 643.200.501. Astro-WISE is an on-going project which started from a FP5 RTD programme funded by the EC Action “Enhancing Access to Research Infrastructures”. The authors thank the anonymous reviewers for their insightful comments, which led to a higher quality and better structure of the paper and indirectly to new research directions.



## References

1. Buddelmeijer, H., Valentijn, E.A.: Automatic optimized discovery, creation and processing of astronomical catalogs. Exp. Astron. (2011). doi:10.1007/s10686-011-9272-z

2. Chaudhuri, S.: An overview of query optimization in relational systems. In: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. PODS '98, pp. 34–43 (1998)
3. Clarke, M.C.: Visualizing boolean operations on a hypercube. Math. Comput. Model. **20**(9), 97–103 (1994)
4. Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM **13**, 377387 (1970)
5. Karnaugh, M.: The map method for synthesis of combinational logic circuits. Trans. AIEE Part I **72**(9), 593–599 (1953)
6. Mwebaze, J., Boxhoorn, D., Valentijn, E.A.: Astro-wise: tracing and using lineage for scientific data processing. In: Proceedings of the 2009 International Conference on Network-Based Information Systems, NBIS 09, pp. 475480. IEEE Computer Society, Washington, DC (2009)
7. Sutcliffe, G.: The TPTP problem library and associated infrastructure. The FOF and CNF parts, v3.5.0. J. Autom. Reason. **43**(4), 337–362 (2009)
8. Venn J.: On the diagrammatic and mechanical representation of propositions and reasonings. Phil. Mag., Series 5 **10**(59), 1–18 (1880)